CC51B - Arquitectura de Computadores : Tarea #4

El ZX Spectrum 48K

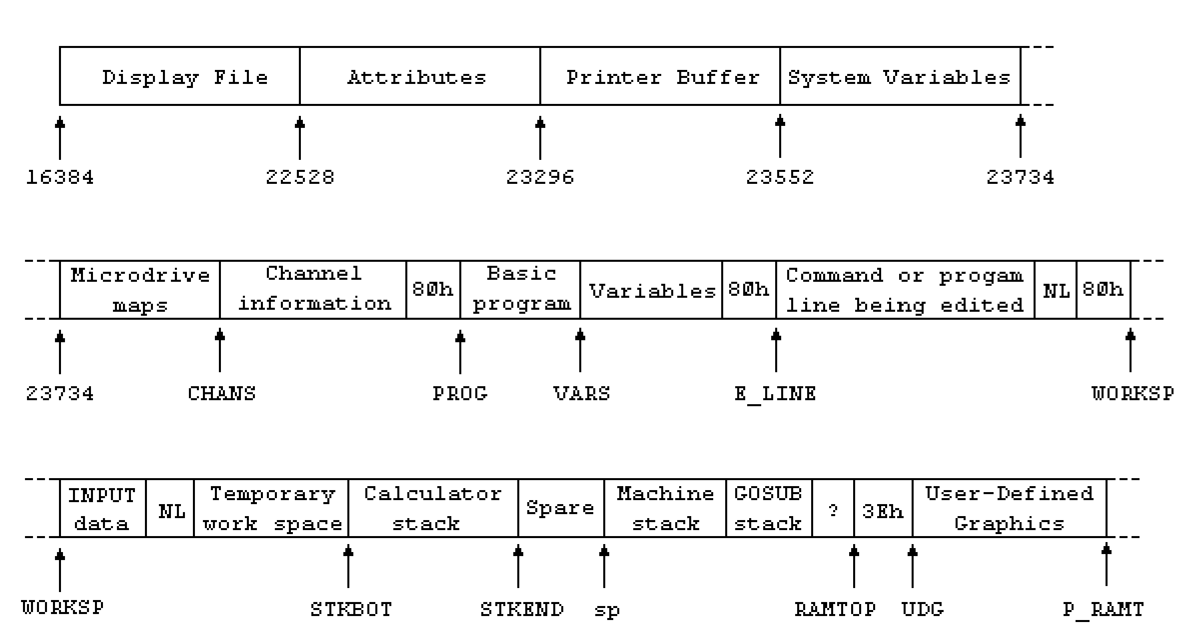
por Denis Fuenzalida

(Introducción / Lenguaje de Máquina / Interrupciones / Memoria / Canales / Punto Flotante)

La memoria

Al contrario de lo que uno esperaría, el ZX Spectrum tiene las direcciones de memoria más bajas destinadas (los primeros 16K) a la ROM, el interprete de BASIC, el juego de caracteres, etc.

La RAM se ubica desde la dirección de memoria 16384 (4000h) hasta la 65535 (FFFFh). La memoria entre las direcciones 16384 y 23296 es usada como la memoria de video (256*192 pixels), donde los primeros 6Kb contienen los bits de la imagen como unos y ceros, y los últimos 768 bytes seleccionan los atributos de color de la imagen, entre 8 colores posibles (pero en total, 256 combinaciones de color, brillo, etc).



Un esquema de la memoria del ZX Spectrum, con algunas zonas delimitadas por *Variables del sistema*

Las direcciones entre la 23296 y la 23552 son usadas como buffer de memoria para la impresora. Ente la 23552 y la 23734 se encuentran las llamadas variables del sistema, que almacenan estados del programa en ejecución, lecturas de ports y punteros a diferentes areas de la memoria.

Los primeros 16K : Memoria *Peleada*

Cuando la ULA está dibujando la pantalla, necesita accesar memoria de video; la RAM no puede ser leida por 2 dispositivos (la ULA y la CPU) al mismo tiempo, por lo que se da más prioridad a la ULA (porque el haz de electrones de la pantalla no puede detenerse) , por lo que los programas que lean los primeros 16K (entre las direcciones

4000h y 7FFFFh) o traten de leer del port #FE (cuando la ULA debe entregar el resultado), serán demorados si la ULA está leyendo la pantalla. Cabe notar que este efecto ocurre solo cuando la pantalla de caracteres se dibuja. Cuando el borde se dubuja, la ULA provee el resultado y ninguna demora sucede. Los detalles precisos son como sigue:

- En el ciclo #14335 (justo uno antes que se alcance el borde sup. izq.) el delay es de 6 ciclos.
- En el ciclo #14336 el delay es de 5 ciclos, y de así en adelante, de acuerdo a la siguiente tabla:

```
Cycle #    Delay
-------    -----
 14335       6 (hasta 14341)
 14336       5 (  "     "  )
 14337       4 (  "     "  )
 14338       3 (  "     "  )
 14339       2 (  "     "  )
 14340       1 (  "     "  )
 14341   No hay delay
 14342   No hay delay
 14343       6 (hasta 14349)
 14344       5 (  "     "  )
 14345       4 (  "     "  )
 14346       3 (  "     "  )
 14347       2 (  "     "  )
 14348       1 (  "     "  )
 14349   No hay delay
 14350   No hay delay
```

y así, hasta el ciclo #14463 (siempre relativo al comienzo de la interrupción), en donde el haz de electrones alcanza el borde otra vez por 96 ciclos más. En el ciclo #14559 (96 ciclos después) la misma situación se repite. Esto es válido para las 192 filas de píxels de la pantalla. Cuando la ULA está actualizando el borde de la pantalla, el delay nunca ocurre.

Cuando se cuentan ciclos, entonces varias cosas deben considerarse en el conteo. Una, es el tiempo del setup de la interrupción; otro es el momento preciso dentro de una instrucción en donde el R/W o la operación de I/O se realiza (ver la tabla más abajo). Una última cosa: el hecho de que una interrupción no puede ocurrir en medio de una instrucción (para esto un `HALT` cuenta como muchos `NOP`s, por lo que algunos ciclos pueden perderse mientras se espera que la instrucción en curso se completa.

Ahora, todo lo que falta es conocer exactamente en que punto(s) dentro de un READ/WRITE o una operación de I/O actuando, conocer donde aplicar el delay. Esto depende para cada instrucción. Para las operaciones de 1 byte que no accesan la RAM ni efectuan acceso a I/O, el unico punto afectado es el *fetch* que ocurre en el primer ciclo de la instrucción, y la dirección para probar si es memoria de los primeros 16K de RAM es el valor actual del program counter (PC)

Por ejemplo, para un `NOP` (4 ciclos), solo el primer ciclo será afectado y solo si el PC está en el primer trozo de la memoria. Por esto, si se ejecuta en memoria *peleada* en el ciclo #14334, ningún delay ocurrirá y la siguiente instrucción será (probablemente)

ejecutada en el ciclo #14338, pero si el `NOP` se ejecuta en el ciclo #14335, será demorada por 6 ciclos, por lo que tomará 6+4=10 ciclos para que la siguiente instrucción pueda ser (probablemente) ejecutada en el ciclo #14345. Este caso será anotado en la tabla a continuación como `pc:4`, lo que quiere decir que si el PC queda dentro de los primeros 16K de la RAM, entonces el primer ciclo será demorado y los 3 restantes estarán libres de delays.

El "probablemente" del párrafo anterior es poque, a menos que el `NOP` esté en PC=32767 (justo antes de los siguientes 16K de RAM), la siguiente instrucción será sujeta a otro delay cuando se haga su fetch (el primer ciclo de una instrucción es siempre sujeto a delays), debido a que el número de ciclo relativo al comienzo del frame también ha sido demorado.

Una fila como `HL+1:3` significa que si HL+1 está en el rango entre 16384-32767 (memoria peleada), y el ciclo en ejecución es sujeto de delays, entonces el delay correspondiente al ciclo actual será insertado antes del número de T-states que aparecen despues del **:**

Las cosas se ponen un poco más dificiles con las instrucciones de más de un byte. El siguiente es un poco de pseudo-código que aplicaa los delays a una instrucción con una lista en la tabla que se lee como '`pc:4, hl:3`' (e.g. `LD (HL),A`):

Things get a bit more difficult with more-than-one-byte-long instructions. Here's the sample pseudocode to apply delays to an instruction with an entry in the table which reads '`pc:4,hl:3`' (e.g. `LD (HL),A`):

```
If (16384 <= PC <= 32767) {
        Insertar el delay que corresponda al ciclo actual,
        relativo al comienzo del frame;
        }

        Delay por 4 ciclos (tiempo después de 'pc:').

If (16384 <= PC <= 32767) {
        Insertar el delay correspondiente al ciclo actual
        }

Hago el STORE del acumulador A en (HL)

Delay por 3 ciclos (que es lo que toma el STORE)
```

Ejemplo 1: Si el PC = 25000 y HL = 26000 y la instrucción en #25000 es `LD (HL),A`, y estamos en el ciclo #14335:

- Inserto 6 ciclos (contando desde el #14335), yendo hasta el #14341.
- Leemos el op.code
- Insertamos 4 ciclos (op.code fetch). Estamos en el ciclo #14345.
- Insertamos 4 ciclos (contando desde el #14345). Estamos en el #14349.
- Hacemos el STORE del byte.
- Insertamos 3 ciclos más (escribiendo en [HL])

El próximo op.code será leido en el ciclo #14352 (y se insertarán 5 ciclos , debido a que PC=25001).

Ejemplo 2: lo anterior, pero ahora PC=40000 (memoria 'superior', no peleada):

- Se lee el op.code
- Insertamos 4 ciclos (fetch del op.code). Ahora estamos en el ciclo #14339.
- Insertamos 2 ciclos (contando desde el #14339), con lo que estamos en el #14341.
- Guardamos el byte.
- Insertamos 3 ciclos (para escribir en [HL]).

Si un elemento en la tabla tiene algo como ' `io:5` ', lo que quiere decir que si el port de I/O es par (bit 0 = 0, como en el port FEh) entonces se toma como una dirección en los primeros 16K (memoria peleada).

Los valores para los registros listados en la tabla son relativos al valor inicial del registro cuando la instrucción va a ejecutarse.

Leyenda :

- `dd` es cualquiera de los registros BC,DE,HL,SP
- `qq` es cualquiera de los registros BC,DE,HL,AF
- `ss` es cualquiera de los registros BC,DE,HL
- `cc` es cualquier condición (si es aplicable): NZ,Z,NC,C,PO,PE,P,M
- `nn` es un número de 16 bits
- `n` es un número de 8 bits
- `b` es un número entre 0 y 7 (instrucciones `BIT`/`SET`/`RES`)
- `r` y `r'` es cualquiera de los registros A,B,C,D,E,H,L
- `alo` es una operación lógica o aritmética: `ADD`/`ADC`/`SUB`/`SBC`/`AND`/`XOR`/`OR`/`CP`
- `sro` es un shift o rotación de bits: `RLC`/`RRC`/`RL`/`RR`/`SLA`/`SRA`/`SRL` and `SLL` (no documentada)

Para las instrucciones condicionales, las listadas entre corchetes (**{}**) significan que solo se realizarán si la condición se cumple. Si la instrucción es no-condicional (e.g. `CALL nn`), lo listado entre corchetes debe ignorarse.

Los prefijos CB/ED/DD/FD cuentan SIEMPRE como un `pc:4`. Esto no se contará en cada instrucción. También, en lugares donde un aparezcan instrucciones que involucren HL, asumiremos que serán reemplazados por IX o IY (y lo mismo para las H y L por separado), cuando sean válidas. El timing para las instrucciones con un operando de la forma (IX/IY+n) no ha sido testeado exahustivamente.

En algunas operaciones del tipo read-modify-write (como `INC (HL)`, entonces la operación de write es siempre la última. Puede ser importante en qué punto exacto se actualiza la imagen, por ejemplo. En esos casos, ese punto será anotado para mayor claridad como "(write)" después de esa dirección.

```
Instrucción    Breakdown
-----------    ---------
```

```
NOP;            pc:4
CB prefix;
ED prefix;
DD prefix;
FD prefix;
LD r,r';
alo A,r;
sro r;
BIT b,r;
SET b,r;
RES b,r;
INC/DEC r;
EXX;
EX AF,AF';
EX DE,HL;
DAA;
CPL;
NEG;
IM 0/1/2;
CCF;
SCF;
DI;
EI;
RLA;
RRA;
RLCA;
RRCA;
JP (HL)

LD A,I;         pc:5
LD A,R;
LD I,A;
LD R,A

INC/DEC dd;     pc:6
LD SP,HL

ADD HL,dd;      pc:11
ADC HL,dd;
SBC HL,dd

LD r,n;         pc:4,pc+1:3
alo A,n

LD r,(ss);      pc:4,ss:3
LD (ss),r

alo A,(HL)      pc:4,hl:3

BIT b,(HL)      pc:4,hl:4

LD dd,nn;       pc:4,pc+1:3,pc+2:3
JP nn;
JP cc,nn

LD (HL),n       pc:4,pc+1:3,hl:3

LD A,(nn);      pc:4,pc+1:3,pc+2:3,nn:3
LD (nn),A

LD dd,(nn);     pc:4,pc+1:3,pc+2:3,nn:3,nn+1:3
```

```
    LD (nn),dd

    INC/DEC (HL);  pc:4,hl:4,hl(write):3
    SET b,(HL);
    RES b,(HL);
    sro (HL)

    POP dd;        pc:4,sp:3,sp+1:3
    RET;
    RETI;
    RETN

    RET cc         pc:5,{sp:3,sp+1:3}

    PUSH dd;       pc:5,sp-1:3,sp-2:3
    RST n

    CALL nn;       pc:4,pc+1:3,pc+2:3,{pc+2:1,
    CALL cc,nn     sp-1:3,sp-2:3}

    JR n;          pc:4,pc+1:3,{pc+1:1,pc+1:1,pc+1:1,
    JR cc,n        pc+1:1,pc+1:1}

    DJNZ n         pc:5,pc+1:3,{pc+1:1,pc+1:1,pc+1:1,
                   pc+1:1,pc+1:1}

    RLD;           pc:4,hl:7,hl(write):3
    RRD

    IN A,(n);      pc:4,pc+1:4,io:3
    OUT (n),A

    IN r,(C);      pc:5,io:3
    OUT (C),r

    EX (SP),HL     pc:4,sp:3,sp+1:4,sp(write):3,sp+1(write):5

    LDI/LDIR;      pc:4,hl:3,de:3,de:1,de:1,{de:1,de:1,de:1,
    LDD/LDDR       de:1,de:1}

    CPI/CPIR;      pc:4,hl:3,hl:1,hl:1,hl:1,hl:1,hl:1,{hl:1,
    CPD/CPDR       hl:1,hl:1,hl:1,hl:1}

    INI/INIR;      pc:6,io:3,hl:3,{hl:1,hl:1,hl:1,hl:1,hl:1}
    IND/INDR

    Nota: Para las siguientes instrucciones no hay mucha claridad
          debido a su complejidad.

    OUTI/OTIR;     si es la última aparición o una versión sin
repeticiones:
    OUTD/OTDR      pc:5,hl:4,io:3
                   (para las versiones con repeticion)
                   pc:5,hl:4,io:1,pc+1:1,pc+1:1,pc+1:1,pc+1:1,
                   pc+1:1,pc+1:1,pc:1
```
